SPATIAL OVERLAY WITH INEXACT NUMERICAL DATA

David Pullar
Surveying Engineering Department
National Center for Geographical Information and Analysis
University of Maine,
Orono, Maine, U.S.A.

Abstract

A methodology for the operation of spatial overlay is presented in this paper. A general framework for spatial overlay based on concepts in epsilon geometry is developed to cope with the problems of computational errors and handling inaccurate numerical data. These problems normally cause topological inconsistencies and generate spurious effects in the result. A mapping is defined to accommodate the edges and vertices in all spatial layers so they are unambiguously aligned within a prescribed tolerance. Geometrical arguments are given to show the correctness of this approach.

1. Introduction

Spatial overlay is an analytical tool used to integrate multiple thematic layers (coverages) into a single composite layer. Each layer is organized into a polygon-network structure where polygons are assigned to nominal, or ordinal, categories. The data may be efficiently stored using a topological data structure [Peucker and Chrisman 1975]. Spatial overlay has proven to be a very powerful tool to analyse the association among different spatial patterns and land characteristics. Despite its popularity there is practically no theory to guide the development of algorithms.

Development of programs for spatial overlay have been hindered by a number of problems related to map accuracy and computational geometry. If the input coverages are overlaid exactly, then sliver polygons are produced along different versions of the same boundary or spatially correlated boundaries represented in different layers. Slivers are an undesirable byproduct of overlay as they are meaningless and degrade further analysis and interpretation of the data [Goodchild 1978]. Other problems that arise are as follows;

i) repeated application of tolerance intersections cause objects to move outside their tolerance [White 1978],
ii) numerical instability causes topological inconsistencies [Franklin 1984],
iii) spurious affects from fuzzy creep and subtolerent segments [Guevara 1985].

One of the most significant achievements in polygon overlay software can be found in the ODYSSEY system for geographic information processing [Dougenik 1979][Chrisman 1983]. The overlay program for ODYSSEY addressed many of the problems listed above, and computed *tolerant intersections* as a solution to the sliver polygon problem [White 1978]. The primitive operation for intersection was broadened to include a tolerance parameter, and clusters of intersection points were analysed to form consistent nodes [Chrisman 1983]. The value for the tolerance parameter is related

to the accuracy and scale of the input coverages. However, a problem arises when trying to overlay many coverages which have multiple accuracies. A central tenet of this paper is that the concepts and approaches used to analyze spatial data can profit from further improvement of the overlay algorithm. A general framework for map overlay which will integrate many coverages, with multiple tolerances, in a single operation is described. The overlay algorithm is similar to Zhang and Tulip [1990] in avoiding slivers by snapping less accurate points to more accurate points and not moving any point outside its tolerance. We extend this work by presenting a verifiable methodology for the overlay operation.

The proposed approach, called *map accommodation*, is based upon a simple concept of accommodating the geometry between each layer to bring them into alignment in the composite layer. Map accommodation detects and reports all types of intersections and proximities between spatial data, and then objectively analyses the data to resolve conflicts.

An outline for the paper is as follows. The next section describes issues related to geometrical intersection. Section 3 describes an algorithm for reliable polygon set operations. It is a robust algorithm, but it does not place an overall bound on positional errors introduced in the process. Section 4 gives a brief outline of the solution we propose, it aims to both bound and minimize any positional errors. Sections 5 and 6 give a more formal treatment of the overlay problem and the correctness of the proposed solution. Section 7 describes a clustering algorithm which is central to our approach. Section 8 analyses the performance of the proposed algorithm and suggests some enhancements.

2. Geometric Operations

Geometric operations on objects representing physical phenomena pose special problems for the design of computer algorithms. Hoffmann [1989] says that "practical implementations of geometric modeling operations remain error-prone, and a goal of implementing correct, efficient, and robust systems for carrying them out has not yet been attained".

Geometric operations encounter two types of errors; i) numerical errors, and ii) representational errors.

2.1 Numerical Errors

Geometric operations on polyhedral objects represented by floating-point numbers will introduce numerical errors in the result [Hoffmann 1989]. The most common numerical error is round-off error. Geometric operations use intermediate results from numerical computations to derive symbolic information. For instance, when a computed variable is less than, equal to, or greater than zero indicates if a point lies below, on, or above a line. If the difference between the values compared is less than a certain threshold ε, computations can lead to misleading results. Two broad approaches are proposed for treating numerical errors; a) compute an exact result by performing intermediate computations with a higher precision [Ottmann et al 1987], or b) determine error

intervals around geometric objects and perturb a version of the input data so objects are unambiguously related to one another within their respective intervals [Hoffmann et al 1988].

2.2 Representational Errors

The coordinate descriptions for geometric objects, whether explicitly stated or not, are expressions of measurement and include some positional uncertainty. Geometric operations need to, a) capture the notion of "approximate tests", and b) to provide estimates on the accuracy of objects. Guibas and others [1989] describe a general framework for coping with errors in geometric operations called *epsilon geometry*, and show how epsilon-predicates are incorporated in geometric tests.

In a similar vein Milenkovic [1989] describes a technique, called *data normalization,* to perform reliable geometrical set operations on polyhedral objects. Data normalization acts as a preprocessing stage to resolve any topological ambiguities before performing set operations. A version of the input data is perturbed slightly to get better agreement between vertices and edges in the output overlay. The decisions on what to perturb and by how much can become very complex for arrangements of line segments.

Epsilon geometry bears some resemblance to techniques used in computer-assisted cartography. Blakemore [1984] suggested an epsilon band could represent the positional uncertainty of a digitized line, and illustrated its use to answer a point-in-polygon query. It has also been used operationally in procedures for map overlay and feature generalization. The overlay program in ODYSSEY was the first to include an epsilon tolerance to control moving the location of boundaries for the removal of sliver polygons [Dougenik 1979]. In feature generalization an epsilon tolerance is used for line filtering [Chrisman 1983] [Perkal 1966].

The approach used in epsilon geometry can cope with inaccuracies significantly larger than those introduced by numerical errors. Therefore, epsilon geometry provides a good general framework to deal with both numerical and representational errors. This chapter builds on these and other works to explore the use of epsilon geometry in polygon overlay. The next section describes the method used to compute the intersection of polyhedral objects which applies the concepts of data normalization. It is a robust algorithm, but has undesirable drawbacks we wish to improve upon.

3. Data Normalization

This section describes a published algorithm that includes some discussion of the reasoning steps involved in geometric operations. Milenkovic [1989] describes a simple method to give a definite and correct answer to geometric operations on polyhedral objects. The idea is to perturb the positions of objects, that are within a certain threshold ε, so they coincide exactly. The method is called *data normalization*, and it assures all data objects satisfy two numerical tests:

1) no two vertices are closer than a tolerance ε, and
2) no vertex is closer to an edge than a tolerance ε.

Two primitive operations are applied to the data to satisfy these conditions, *vertex shifting* and *edge cracking*. See figures 1 and 2. Vertex shifting will move one vertex to another if they are closer than tolerance ε. If a vertex is within tolerance ε to an edge, edge cracking will move the vertex to a new cracked point along the edge.

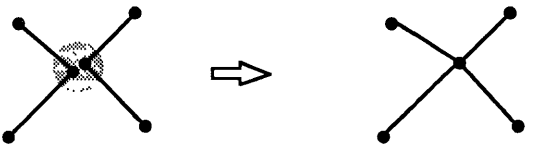


Figure 1: Vertex Shifting

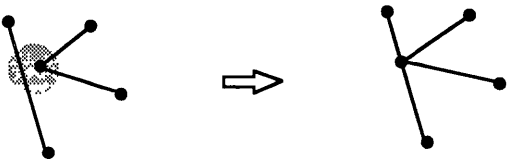


Figure 2: Edge Cracking

The algorithm described by Milenkovic initially makes two passes through the data. The first pass tests for near coincidence between vertices from the input polygons. When two vertices are found within the threshold tolerance then one vertex is shifted and identified with the other vertex. The second pass tests for the proximity of vertices to edges for the input polygons, and does edge cracking where necessary. Edge cracking may introduce further near coincidences. Hence, the algorithm is reiterated until both the numerical tests for data normalization are satisfied. With each iteration slight perturbations accumulate and may lead to positional alterations larger than ε, this is called *creep*. For example, in the left diagram of Figure 3 the vertices u_1 and v_1 are within tolerance ε of edge (u_0v_0). The right diagram shows the results after edge cracking, now the vertices u_2 and v_2 are within ε of the new edge (u_1v_1) which in turn calls for further cracking. Edge cracking may continue in a cascading fashion so that points along edge (u_0v_0) will migrate outside their given tolerance.

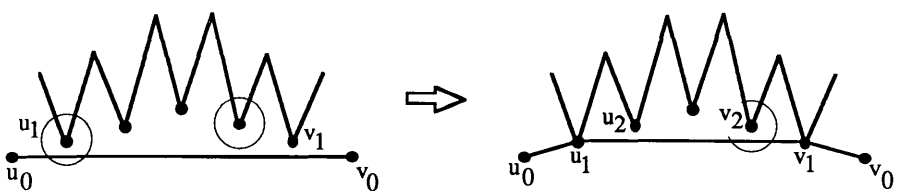


Figure 3: Creep introduced by edge cracking.

This algorithm will compute output polygons with a valid planar topology, so in this sense it is robust. However, positional error will accumulate with each iteration so the

procedure cannot place a constant bound on the extent polygons are perturbed. The next section describes another approach which avoids creep.

4. Proposed Solution

This paper has used the ideas behind data normalization and adapted them to a new approach to overlay called *map accommodation*. This section gives a brief description of the mechanics of our approach to map overlay, and shows how a clustering procedure is incorporated with the primitive operations for vertex shifting and edge cracking. Latter sections will give a more formal and detailed description.

4.1 Issues

The two drawbacks to the algorithm described in the previous section are;

i) the perturbations are performed in an arbitrary fashion, and

ii) it does not prevent creep.

First, there is no objective evaluation of relations between the geometric primitives (vertices and edges) to guide what gets snapped to what. It is a greedy algorithm which accommodates primitives of one polygon to another as violations to normalization conditions occur. We propose an algorithm which objectively evaluates the proximities between geometric primitives to guide the accommodation process. Like the overlay program in ODYSSEY [Dougenik 1979], a clustering strategy is used to minimize the perturbations to the data. The benefit of this approach is it promotes a stable map topology.

The stability of map topology and its relation to a discrete surface model is investigated by Saalfeld [1987]. Measures of stability, or robustness in the topological structure, are defined in terms of a geometrical measure of the closeness of primitives. We propose a strategy to cluster primitives based upon proximity, such that primitives are clumped together to minimize perturbations and provide the greatest separability between cluster centers. In this way we expect greater stability in the overlay transformation.

Second, primitives can migrate from their original locations by a significant distance for certain degenerate configurations. This was discussed briefly in the previous section and, Milenkovic demonstrates a case where a valid polygon can collapse to a point. To avoid the effects of creep we perform clustering in a special way to bound the perturbations.

4.2 Brief Outline of Algorithm

The remainder of this section gives a simple description of map accommodation, and latter sections will go into greater detail. The accommodation algorithm accepts as input N layers of data structured in an topological format. To start, we check all vertex-vertex proximities. If any two vertices are within a geometrical tolerance to one another they are reported in a list. Cluster analysis is performed on the elements in this list to determine consistent output vertices. By consistent we mean vertices satisfy the normalization condition, that is; i) no two vertices are closer than the tolerance, and ii) no vertex is moved greater than its tolerance. In effect, we have performed the task of

vertex shifting to accommodate the vertices among the layers.

Next, we check all vertex-edge proximities. If a vertex is found to be within the given geometrical tolerance to another edge, then the vertex and its closest point along the edge are reported in a list. Cluster analysis is again performed on this list to determine consistent output vertices. Any internal points to edges that are clustered to other points are treated as a cracked edge.

Finally, we check all edge-edge intersections. The composite layer may now be assembled in a straight-forward manner.

4.3 Example

The process just described is illustrated by overlaying two simple geometric objects shown in figure 4.

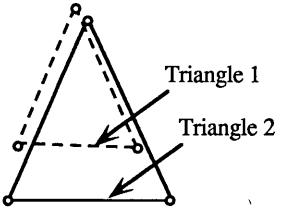


Figure 4: Two figures to be overlaid.

In figure 5, the highlighted vertices are found to be within the geometrical tolerance to one another, and are clustered and identified with a single vertex. In figure 6, the highlighted areas show edges found to be within the tolerance to a vertex. The edges are broken at the closest point to the offending vertex, this point and the vertex are clustered and identified with a single vertex. Now the geometry for the two triangles are accommodated to common vertices and edges. It is a now a relatively simple problem to compute their Boolean intersection. From the resulting figure we can answer approximate tests concerning the coincidence or inclusion of vertices and edges.

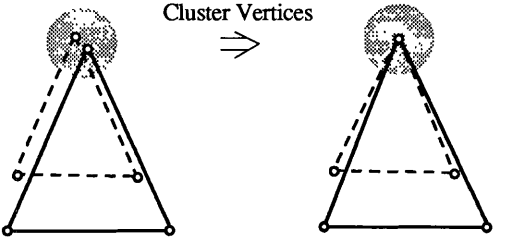


Figure 5: Vertex Shifting

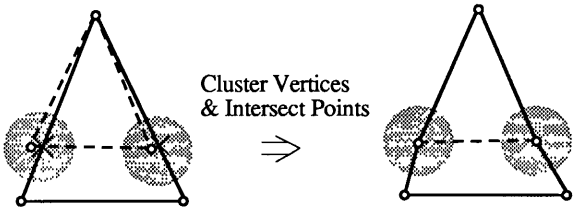


Figure 6: Edge cracking

This section has presented a very simple description of map accommodation. Subsequent sections will give a more formal and detailed treatment of the process.

5 Accommodation Conditions

This chapter is mainly concerned with how the overlay transformation affects the geometry of objects. Map accommodation is required to satisfy five geometric conditions which validate the result of polygon overlay. These conditions form the basis for the design of the map accommodation algorithm.

The basic geometric primitives are vertices and edges, and an associated tolerance parameter called epsilon. A map layer is denoted by the triple $\{V,E,\in\}$, where;

V is the set of vertices v representing points in the plane,

E is the set of edges e made up of ordered pairs of vertices, and

$\in$ is the epsilon parameter ε associated with the vertices or edges.

To establish a convention, we will use lower case symbols to denote primitives of a set and upper case to denote the sets. For example, e_i^R is the i-th element in the R-th set of edges, or in other words $e_i^R \in E^R$. An element e_i^R has an associated tolerance denoted ε_i^R.

A vital part of the overlay process is to accommodate the geometric primitives from all input layers to resolve topological ambiguities. A number of input layers will be mapped to a single composite layer. If a primitive is not within the epsilon tolerance to another primitive then it remain unchanged. However, if it is necessary to accommodate primitives from different layers then this may cause the combining of primitives or the insertion of new primitives. The five conditions for map accommodation determine what sort of changes are allowed. These conditions examine spatial proximities and are defined in terms of the Euclidean distances *d* between primitives. Formally, we define the *accommodation mapping* for 1..N input layers to one composite layer as,

$$\{V^1,E^1,\in^1\}, \{V^2,E^2,\in^2\}, \ldots \{V^N,E^N,\in^N\} \Longrightarrow \{V',E',\in'\}$$

such that the following accommodation conditions are satisfied;

A1) v_i^R is moved to v_j' iff $d(v_i^R, v_j') < \varepsilon_i^R$,

A2) for any two v_i', v_j' implies $d(v_i', v_j') > \text{minimum}(\varepsilon_i', \varepsilon_j')$,

A3) a cracked point p on e_i^R is moved to v_j' iff $d(p, v_j') < \varepsilon_i'$,

A4) for any v_i' and point p on any e_j' implies $d(p, v_i') > \text{minimum}(\varepsilon_i', \varepsilon_j')$,

A5) no two e´ intersect except at a common vertex v´.

An infinite number of mappings will satisfy these condition. Therefore, another constraint is imposed to minimize any positional alterations. This is achieved by using cluster analysis to objectively chose the output vertices to which other vertices are moved. The next section describes the way cluster analysis is used in map accommodation and presents informal arguments to show how the above conditions are satisfied.

6 Accommodation Process

Central to the accommodation process is the clustering procedure. Clustering will analyze points and replace the points which are agglomerated with their weighted centroid. The weight for a point is related to its associated epsilon parameter. For now, we only define the properties of the clustering method and leave the description of the clustering algorithm for the next section. The input to cluster analysis is a set $\{P, \in\}$, where;

P is a set of points p in the plane, and

$\in$ is the epsilon parameter ε associated with each point.

Formally, clustering is a mapping between sets as,

$$\{P, \in\} \Longrightarrow \{P', \in'\},$$

where p_j' is the weighted centroid for agglomerated points $p_i \in P$, and ε_j' is selected as the minimum of the $\varepsilon_i \in \in$, and such that the following clustering properties hold;

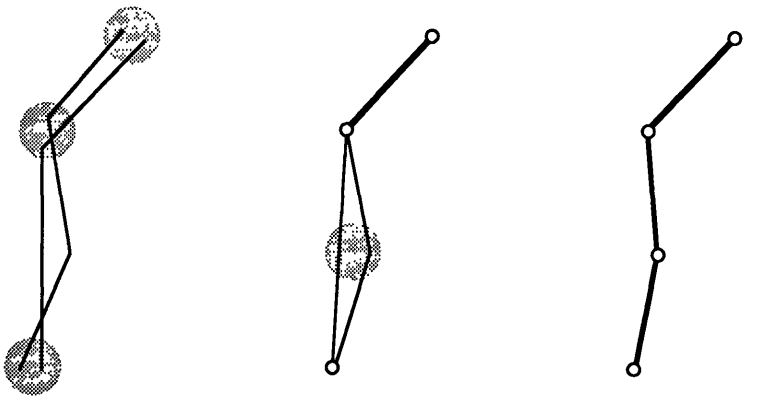
C1) for any p_i clustered to p_j' implies $d(p_i, p_j') < \varepsilon_i$, and

C2) for any two p_i', p_j' implies $d(p_i', p_j') > \text{minimum}(\varepsilon_i', \varepsilon_j')$.

Based on the given properties of a clustering, we describe each stage in the accommodation technique and give informal arguments to show they satisfy the accommodation conditions. This is followed by an outline of each step in the algorithm. The three major stages of the accommodation technique are;

1. Vertex shifting,
2. Edge cracking,
3. Edge intersection.

The order of execution for each stage is designed to detect and resolve correlation between boundaries. Correlated boundaries will demonstrate a similar pattern of curvature. This correlation will be most prevalent at corresponding terminal points and break points along boundary lines. Therefore, detection of coincident vertices should proceed first. Figure 7(a) shows two correlated lines with areas of vertex coincidence highlighted. Figure 7(b) shows the result after vertex shifting. Subsequent detection of vertex to edge proximities will detect correlation along the boundary line. Figure 7(b) also highlights areas of vertex to edge coincidence, and figure 7(c) shows the result after edge cracking. The final stage will detect clear cases for two edges crossing.



(a) Detect vertex coincidence (b) After vertex shifting (c) After edge cracking

Figure 7: Map accommodation for two lines

6.1 Vertex Shifting

The task of vertex shifting is to detect vertices that are approximately coincident, and then compute consistent output vertices. One pass is made through the input data performing pairwise comparisons between vertices in each layer. When the distance between two vertices is discovered to be less than the sum of their tolerances they are reported in a list. This list serves as input to the clustering algorithm. Clusters are computed and any affected vertices are identified with a new vertex at the respective cluster centroid.

Clustering will satisfy properties (C1) and (C2), which is sufficient to prove that conditions (A1) and (A2) for the accommodation mapping (given in §5) are satisfied. That is, no vertex is perturbed outside it epsilon tolerance, and vertices are separated by at least the minimum epsilon tolerance.

6.2 Edge Cracking

Edge cracking detects vertices that lie approximately on an edge, and then computes consistent output vertices which are inserted along the appropriate edge. One pass through the data is required to find all vertices near edges. A vertex v_i is considered near to an edge e_j, by first finding point p as the orthogonal projection of v_i onto e_j, when $d(v_i,p) < (\varepsilon_i+\varepsilon_j)$. Any affected vertices and cracked edges are reported in the

list.

There does exist degenerate cases that require additional checking. In certain geometric configurations cracking an edge will cause further edge cracking. Figure 8 shows such a case, v causes e_1 to be cracked at p_1 which in turn causes e_2 to be cracked at p_2. The later point is called an *induced intersect point,* these constructs were first identified in the overlay part of the ODYSSEY program [Harvard 1983].

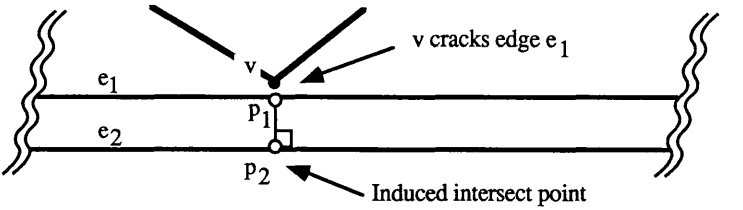


Figure 8: Edge e_1 is cracked at p_1 which may cause edge e_2 to be cracked at p_2

To guard against these degenerate cases requires an additional test. All new vertices located along a cracked edge are tested against all other edges. If any induced intersect points are discovered they are reported in the list. This list serves as input to the clustering algorithm. Clusters are computed and any affected vertices are identified with a new vertex at the respective cluster centroid.

Clustering properties (C1) and (C2) are sufficient to guarantee conditions (A1) and (A3) for the accommodation mapping (given in §5) are satisfied. Edge cracking will not violate condition (A2) because the only way for two vertices to move close to each other is if they are cracked by an edge between them, and therefore they must already have been discovered and evaluated in the edge cracking procedure. By searching for and including induced intersect points in the cluster analysis will guarantee there are no further violations to condition (A4). Therefore, conditions (A1) to (A4) for the accommodation mapping are satisfied.

6.3 Edge Intersection

Edge intersection identifies a common vertex at the point where two or more edges cross. One pass through the data is required to find all cases where edges cross at internal points. Note all intersections are computed without ambiguities since vertices and edges now satisfy conditions (A1) to (A4). However, by creating a new vertex at the intersection point may cause a violation to condition (A4). Figure 9 illustrates the degenerate case that needs to be treated. An additional test for further edge cracking is required (with the new intersection points only) to detect induced intersections. Again, all intersection points and induced intersect points serve as input to cluster analysis.

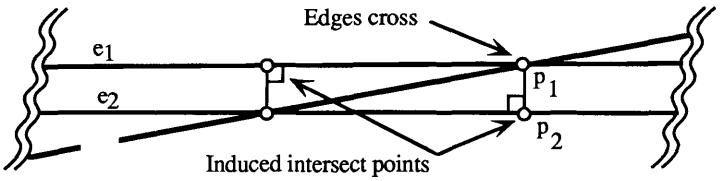


Figure 9: Edges e_1 and e_2 intersect at p_1, which may cause edge e_3 to be cracked at p_2

When two edges cross they are cracked at a common point. This point is identified as a common vertex for both edges, so condition (A5) of the accommodation mapping is now satisfied. After this, all the conditions for the accommodation mapping are satisfied and we can proceed to rebuild the topological structure for the composite layer in a reasonably straight forward manner by tracing polygonal paths.

6.4 Accommodation algorithm

An outline of the algorithm for accommodation is presented. Accommodation calls the cluster analysis procedure. For simplicity it is assumed clustering will satisfy conditions (C1) and (C2) for any input, and then details for the clustering algorithm are explained in the next section.

PRELIMINARY

The algorithm accepts as input any number of layers, each composed of a set of edge-paths. A tolerance parameter is associated with each edge, therefore tolerances may also vary within layers. For convenience we shall denote the R-th layer in our set terminology as $\{V^R,E^R,\epsilon^R\}$. The algorithm needs a data structure to store intersect points to be clustered, this is called MSET.

ALGORITHM - overlay of N layers of chains

Map-accommodation: $\{V^1,E^1,\epsilon^1\}, \{V^2,E^2,\epsilon^2\}..\{V^N,E^N,\epsilon^N\} \Longrightarrow \{V',E',\epsilon'\}$

Step 1. Do pairwise comparison of vertices v_i^R, v_j^S where R≠S, and report all pairs within the tolerance, i.e. $d(v_i^R, v_j^S) < (\varepsilon_i^R + \varepsilon_j^S)$, to the set MSET.

Step 2. Perform a clustering on the points in MSET. Identify all agglomerated vertices v with the appropriate cluster centroid v′ and assign the minimum ε to ε'.

Step 3. Do a pairwise comparison of vertices and edges v_i^R, e_j^S where R≠S, and report pairs within the tolerance, i.e. let p be point on e_j^S perpendicular to v_i then $d(v_i^R,p) < (\varepsilon_i^R + \varepsilon_j^S)$, to the set MSET.

Step 4. Repeat Step 3 for the newly cracked points, i.e. the cracked points p on e_j^S, to find any induced intersect points, and report these pairs to the set MSET.

Step 5. Perform a clustering on the points in MSET. Identify all agglomerated vertices v and cracked points on e with the appropriate cluster centroid v′ and assign the minimum ε to ε'.

Step 6. Do pairwise intersection of edges e_i^R, e_j^S where R≠S, and report pairs that intersect (at an interior point on both edges) to the set MSET.

Step 7. Repeat Step 3 for intersect points, i.e. the point p on e_i^R and e_j^S, to find any induced intersect points, and report these pairs to the set MSET.

Step 8. Perform a clustering on the points in MSET. Identify all intersect points on e with the appropriate cluster centroid v′ and assign the minimum ε to ε'.

7. Clustering

A central part of our approach to the accommodation process is the clustering algorithm. The previous section defined the properties of a clustering, this section describes how clustering is performed. A clustering problem is defined as a partition of a finite set into n disjoint sets based upon minimizing an objective function. The objective function is typically some proximity measure to bring out intrinsic structure in the data. The complexity of obtaining a global optimal solution is NP-complete for n-partitions (n>2) in two or more dimensions [Brucker 1978]. This is not computationaly feasible, so a suboptimal heuristic solution is proposed.

The task at hand is to describe; i) the objective function used to measure proximity between clusters, and ii) the strategy used to form partitions of the data.

7.1 Proximity Matrix

The proximity matrix represents an index of similarity (or dissimilarity) measures between pairs of clusters. The most commonly used criteria for computing these similarity measures is a *square-error criteria,* and is based on minimizing the square error between component points and their computed cluster centroid [Jain & Dubes 1988]. This is similar to minimizing the within-cluster variation and maximizing the between-cluster variation. We show how this criteria is adapted to clustering points with a geometrical tolerance.

The weighted arithmetic mean for a set of values x_i and their associated weights w_i is;

$$\bar{a}_w = \frac{\sum_{i=1}^{N} a_i w_i}{\sum_{i=1}^{N} w_i} .$$

The weighted mean vector for a cluster K, denoted m^k, is defined as the *cluster centroid.* This is computed by the weighted arithmetic mean for ordinates of the cluster points. If a coordinate is denoted by the pair {x,y} then the weighted coordinate centroid m^k is denoted as $\{\bar{x}_w, \bar{y}_w\}$. The weight is related to the positional uncertainty associated with a point, and is defined as the inverse of the square of the epsilon tolerance, i.e. $w = 1/\varepsilon^2$.

The square-error for the k^{th} cluster with n_k members is given as;

$$e_k^2 = \sum_{i=1}^{n_k} \left(x_i^k - m^k\right)^T \left(x_i^k - m^k\right) = \sum_{i=1,n_k} d(x_i^k, m^k)^2 .$$

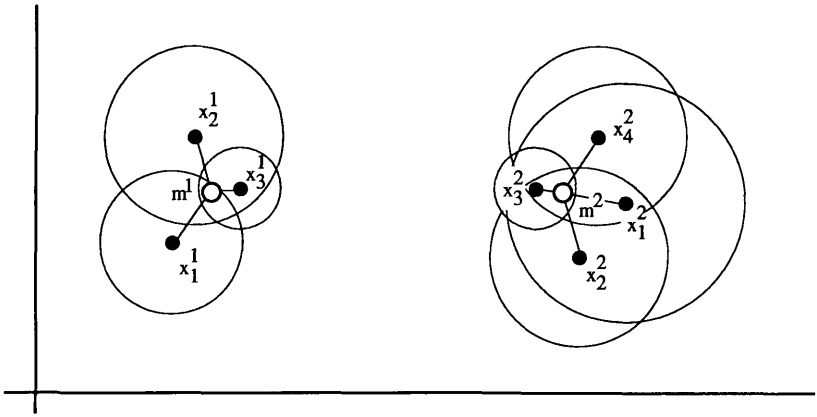


Figure 10: Distances used in computing square-error

A minimum variance partition is defined as a clustering which minimizes the sum of the square-error for a fixed number of N clusters, that is by minimizing the expression;

$$E_N^2 = \Sigma_{k=1,N}\ e_k^2 .$$

The similarity measure computed in this chapter must additionally obey the following constraints;

1) the distance between a cluster centroid and all its member points is not greater than the given geometrical tolerance, and
2) the function should seek to minimize the number of clusters.

To fulfill the first requirement, we define the *bounded-square-error* for the k^{th} cluster C_k as;

$$\bar{e}_k^2 = e_k^2 \quad \text{if } \forall_{x_i \in C_k}\ d(x_i, m^k) \le \varepsilon_i$$
$$= \infty \qquad \text{otherwise,}$$

where ε_i is the geometrical tolerance associated with x_i. This says that we are only interested in points within the given geometrical tolerance to the centroid, otherwise the square-error can be some very large number.

In the second requirement, we need to limit the final number of clusters by gradually merging clusters. The idea is to find the minimum number of clusters which satisfy conditions (C1) and (C2). At the same time the partitioning chosen should yield the minimum value for E_N^2 using a bounded-square-error criteria. A solution to this problem is computationally not feasible. In fact, it requires examining the power set of all points. The next section describes a suboptimal solution to the problem.

7.2 Clustering Strategy

We have already shown that an optimal partitioning based on minimizing an objective function is not computationally feasible. Therefore, a selection strategy is used to reduce the number of partitions evaluated to achieve a "reasonable" approximation. The selection process is designed to converge to a local minima of the objective function. Jain and Dubes [1988] give an extensive discussion on the factors involved in cluster strategies. The major choices are between hierarchical and partitional schemes. Hierarchical clustering schemes organize the data into a nested sequence of groups. Partitional clustering schemes successively determine partitions of clusters such that points are moved between clusters in an effort to improve a criteria function. The major disadvantages we see for a partitional clustering are; i) it is very sensitive to a hill-climbing solution, ii) it is designed to solve for a fixed number of partitions. A hierarchical procedure is not as sensitive to a hill-climbing solution; but as Jain and Dubes state, its most desirable feature is in modeling the global structure of the data.

An agglomerative algorithm for hierarchical clustering is proposed. It starts by placing each point into an individual cluster. A proximity matrix made up of similarity measures between clusters is computed. This matrix is interpreted to merge two or more clusters at each level in the hierarchy. The process is repeated to form a sequence of nested clusters. The bounded-square-error criteria will terminate when all values in the proximity matrix are infinity. This provides a reasonable solution to minimizing the number of clusters.

The algorithm needs an appropriate data structure to store clusters and their respective member points. Operations for merging clusters and for finding which cluster a particular point is in must be supported. A very efficient data structure called a MERGE-FIND ADT is described in Aho, Hopcroft and Ullman [1985] for this purpose.

8. Analysis of Algorithm

We can analyse the computational cost for the accommodation mapping in each of the three steps by examining the complexity for; i) geometrical intersection, and ii) clustering.

First, geometrical intersection involves the pairwise comparison between primitives in the various layers. If there are M layers each having T primitives (interpret this as either the number of vertices or edges) then it requires $(MT)^2$ proximity comparisons between primitives in a brute force approach. The number of points reported will depend on the degree of spatial correlation between primitives in different layers. We estimate the worst case occurs when all the layers are the same giving 2MT points.

A plane sweep solution to geometrical intersection [Preparata and Shamos 1985] is unsuitable because the sweep invariant requires a strict order between edges and the vertical sweep line. A modified sweep technique using a band sweep is used in the ODYSSEY program [White 1978], and this is claimed to work well. An alternative method using a griding technique [Franklin 1989] was adopted for our implementation.

The edges were organized into edge-cell pairs by testing if the band (given by the epsilon tolerance about an edge) overlapped a grid cell. Then a brute force method was used to compute tolerance intersection within each cell. Performance tests on experimental data, which assume a uniform distribution of line segments, show favorable execution times for a grid partitioning technique compared to the plane sweep technique [Pullar 1990].

Second, the computational cost for clustering is extremely high using a brute force approach. Let N=2MT, then it would require N(N-1)/2 computations to construct the proximity matrix for a set of intersect points. To process this matrix for clustering requires $N^2(N-1)/4$ computations in the worst case, i.e. all N points merge to a single cluster. Therefore, the computational complexity of the algorithm is of order $O(N^3)$ in the worst case. Day and Edelsbrunner [1984] offer an improvement on this by efficient determination of nearest neighbors in the clustering algorithm, they describe an algorithm of $O(N^2 \log N)$ in the worst case. This was still felt to be an unacceptable cost.

In our implementation, the efficiency of clustering was improved by incorporating the grid partitioning technique in the algorithm. An alternative clustering procedure, which in principle works the same as an agglomerative algorithm, is used in collaboration with the uniform grid. Assuming points are uniformly distributed over the coverage, a nearest-neighbor search can be localized using a grid superimposd over the data. The technique is described in Murtagh [1983], and an upper-bound for the expected time complexity is reported to be O(N log N). If the grid resolution is no smaller than the maximum epsilon tolerance then a nearest-neighbor search may be localized to the current grid cell and its adjacent group of grid cells. In our experiments the partitioning techniques had a very satisfactory average behavior and exhibited an O(N) cost behavior. Further details of the algorithm will be published in a future article.

9. Conclusion

The main objective of this paper was to develop a methodology for map overlay which overcomes problems of computational errors and handling numerical data of an uncertain pedigree. A technique used to perform reliable geometrical set operation in solid modelling systems, called data normalization, is adapted to the map overlay problem. We show how the proposed technique, called map accommodation, will prevent creep in the geometry of primitives and promote stability in map topology. We have also presented informal arguments which demonstrate the correctness of this approach.

The advantage of our approach is it breaks the complex task of map overlay into simpler tasks which require simple data structures for implementation. The key operations for reporting intersections and spatial proximities between primitives, and clustering points involve a significant computational workload. Therefore, an efficient approach needs to be incorporated in the all stages of the algorithm to provide satisfactory performance. We propose the use of a grid partitioning technique.

Acknowledgements

I would like to express my appreciation to Renato Barrera for the advice and ideas he contributed towards this work. Support for this project is provided by Prime Wild GIS Inc., and from the U.S. National Science Foundation through the NCGIA.

References

Aho A., Hopcroft J. and Ullman J., 1985; *Data Structures and Algorithms*. Addison-Wesley Publishing Co., Reading, MA.

Blakemore M., 1984; *Generalization and Error in Spatial Databases*. Cartographica, Volume 21. pp.131-139

Brucker P., 1978; *On the Complexity of Clustering Problems*. In: Optimierung und Operations Research, Editors R.Hen, B.Korte and W.Olletti, Springer, Berlin.

Chrisman N., 1983; *Epsilon Filtering: A Technique for Automated Scale Changing*. Proceedings 43rd Annual Meeting of ACSM, Washington, DC. pp.322-331

Day W. and Edelsbrunner H., 1984; *Efficient Algorithms for Agglomerative Hierarchical Clustering Methods*. Journal of Classification, 1. pp.7-24

Dougenik J., 1980; *WHIRLPOOL: A program for polygon overlay*. Proceedings Auto-Carto 4, Vol.2, Reston, VA. pp.304-311

Dutton G., 1978; *Harvard Papers on Geographical Information Systems*. Addison-Wesley Publishing Co., Reading, MA.

Franklin W.R., 1984; *Cartographic Errors Symptomatic of Underlying Algebra Problems*. 1st International Symposium on Spatial Data Handling, Zurich. pp.190-208

Franklin W.R., 1989; Uniform Grids: *A Technique for Intersection Detection on Serial and Parallel Machines*. Proceedings of Auto-Carto 9, Baltimore, Maryland.

Goodchild M., 1978; *Statistical Aspects of the Polygon Overlay Problem*. In: Harvard Papers on Geographical Information Systems, Vol.6, Editor G. Dutton 1978.

Guevara J., 1985; *Intersection Problems in Polygon Overlay*. Unpublished paper, available from author through E.S.R.I., Redlands, CA.

Guibas L., Salesin D. and Stolfi J., 1989; *Epsilon Geometry: Building Robust Algorithms from Imprecise Computations*. Proceedings 5th Annual ACM Symposium on Computational Geometry, West Germany.

Harvard, 1983; *WHIRLPOOL Programmer Documentation*. Harvard Computer Graphics Laboratory, Cambridge, MA.

Hoffmann C., 1989; *Geometric and Solid Modeling*. Morgan Kaufmann, San Mateo, CA.

Jain A. and Dubes R., 1988; *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ.

Milenkovec V., 1989; *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*. In: Geometrical Reasoning, Editors D. Kapur and J. Mundy, MIT Press, Cambridge, MA.

Murtagh F., 1983; *Expected-Time Complexity Results for Hierarchic Clustering Algorithms Which Use Cluster Centres*. Information Processing Letters, 16. pp.237-241

Ottmann T., Thiemt G. and Ullrich C., 1987; *Numerical Stability of Geometric Operations*. Proceedings 3rd ACM Symposium on Computational Geometry. Waterloo, pp.119-125

Perkal J., 1966; *An Objective Approach to Map Generalization*. Discussion Paper 10, Ann Arbor MI, Michigan Inter-University Community of Mathematical Geographers.

Peucker T. and Chrisman N., 1975; *Cartographic Data Structures*. American Cartographer, Vol.2. pp.55-69

Preparata F. and Shamos M., 1985; *Computational Geometry*. Springer-Verlag, New York.

Pullar D., 1990; *Comparative Study of Algorithms for Reporting Geometrical Intersections*. 4th International Symposium on Spatial Data Handling, Zurich.

Saalfeld A., 1987; *Stability of Map Topology and Robustness of Map Geometry*. Proceedings of Auto-Carto 8, Baltimore, Maryland.

White D., 1978; *A Design for Polygon Overlay*. In: Harvard Papers on Geographical Information Systems, Vol. 6, Editor G. Dutton 1978.

Zhang G. and Tulip J. 1990; *An Algorithm for the Avoidance of Sliver Polygons and Clusters of Points in Spatial Overlay*. 4th International Symposium on Spatial Data Handling, Zurich. pp.141-150